



Scala & JEE

Ant Kutschera, ITIC 2012
www.maxant.ch

Quick Survey?

- Who uses Scala?
 - Who wants to use Scala?
 - Who thinks it's a waste of time?
 - Who thinks it isn't time yet?
-

Why Scala?

- Functional Programming
 - Support for monads, pattern matching, immutability built into language
 - 3M€ investment into Typesafe
 - Someone is betting on it
 - Concise language
 - Less code, more maintainable?
 - More features
 - Actors, superior collections, parallel collections, DSL support, XML support
 - Fun!
-

Scala Environment

- Everything is new
 - Build tool => sbt
 - Test framework => scalatest, specs2
 - Database Frameworks => squeryl, ScalaQuery / SLICK
 - Web => Lift / Play!
 - I can't leverage my JEE knowledge
 - Maturity of new products?
-

Java Enterprise Edition

- JSF 2 => Primefaces, etc. have full Ajax Support incl. Server Push
 - EJB => Business Tier
 - Web Services (SOAP & REST)
 - JPA
 - JCA for Integration

 - Cross cutting: Security, transactions, resources, concurrency, scalability, reliability, configuration, etc.
-

So let's write an EJB

```
package ch.maxant.scalabook.services
import javax.ejb.Stateless

@Stateless
class DoesNotWorkService {

    @Inject var log: Logger = null
    @PersistenceContext(unitName="theDatabase") var em: EntityManager = null
    @Resource var ctx: SessionContext = null
    @EJB var userService: UserService = null

    def foo(bar: String) = {
        //something useful
    }
}
```

So let's write an EJB

```
package ch.maxant.scalabook.services
import javax.ejb.Stateless

@Stateless
class DoesNotWorkService {

    @Inject var log: Logger = null
    @PersistenceContext(unitName="theDatabase") var em: EntityManager = null
    @Resource var ctx: SessionContext = null
    @EJB var userService: UserService = null

    def foo(bar: String) = {
        //something useful
    }

}
```

- Doesn't work!

org.jboss.as.server.deployment.DeploymentUnitProcessingException: JBAS014544:

No EJB found with interface of type

'ch.maxant.scalabook.services.DoesNotWorkService' for binding
ch.maxant.scalabook.services.DoesNotWorkClient/svc

So let's write an EJB

```
@Stateless
@ScalaSignature(bytes="")
public class DoesNotWorkService
  implements ScalaObject
{
  @Inject
  private Logger log;

  @PersistenceContext(unitName="theDatabase")
  private EntityManager em;

  @Resource
  private SessionContext ctx;

  @EJB
  private UserService userService;
```



Generated decompiled bytecode

Required modification

```
package ch.maxant.scalabook.services
import javax.ejb.Stateless

@Stateless
@LocalBean
class DoesNotWorkService {

  @Inject var log: Logger = null
  @PersistenceContext(unitName="theDatabase") var em: EntityManager = null
  @Resource var ctx: SessionContext = null
  @EJB var userService: UserService = null

  def foo(bar: String) = {
    //something useful
  }
}
```


Business code example

```
def reserveOffer(event: EventOffer, tariffs: Seq[Tarif]) = {  
    var reservations = tariffs.par.map{ t =>  
        getAdapter(t.bookingSystem) match {  
            case Some(a) => a.reserveOffer(event, t)  
            case _ => throw new  
                IllegalStateException(  
                    "Unknown adapter " +  
                    t.bookingSystem)  
        }  
    }  
    //so that the caller isnt surprised, we  
    //should convert this back to a non-parallel sequence  
    reservations.seq  
}
```

- Uses: parallel collections, pattern matching, monadic function
-

More code examples

```
def distinctCities = {  
  getTeasers().map(_ .address.city).distinct.sorted  
}
```

Collections API
(monads)

DSL

```
val premium = forEveryTicketIn(shoppingCart){ ticket =>  
  if( ticket.value > 50.CHF)  
    14 .percent of ticket.value  
  else  
    11 .percent of ticket.value  
} subtract (3.10 .if_total > 10.CHF )
```


More code examples

```
case class WeatherData(
    lat: Double,
    long: Double,
    city: String,
    maxTemperaturePerMonth: List[Double]
){}
```

Simple model class

MapReduce, in parallel

```
val cities = List(
    WeatherData(0.0, 0.0, "Heraklion", List(20.0, 21.1, 23.2, 24.1, 25.6, 27.0, 28.1, 29.2, 30.3, 31.4, 32.5, 33.6, 34.7, 35.8, 36.9, 38.0, 39.1, 40.2, 41.3, 42.4, 43.5, 44.6, 45.7, 46.8, 47.9, 49.0, 50.1, 51.2, 52.3, 53.4, 54.5, 55.6, 56.7, 57.8, 58.9, 60.0, 61.1, 62.2, 63.3, 64.4, 65.5, 66.6, 67.7, 68.8, 69.9, 71.0, 72.1, 73.2, 74.3, 75.4, 76.5, 77.6, 78.7, 79.8, 80.9, 82.0, 83.1, 84.2, 85.3, 86.4, 87.5, 88.6, 89.7, 90.8, 91.9, 93.0, 94.1, 95.2, 96.3, 97.4, 98.5, 99.6, 100.7, 101.8, 102.9, 104.0, 105.1, 106.2, 107.3, 108.4, 109.5, 110.6, 111.7, 112.8, 113.9, 115.0, 116.1, 117.2, 118.3, 119.4, 120.5, 121.6, 122.7, 123.8, 124.9, 126.0, 127.1, 128.2, 129.3, 130.4, 131.5, 132.6, 133.7, 134.8, 135.9, 137.0, 138.1, 139.2, 140.3, 141.4, 142.5, 143.6, 144.7, 145.8, 146.9, 148.0, 149.1, 150.2, 151.3, 152.4, 153.5, 154.6, 155.7, 156.8, 157.9, 159.0, 160.1, 161.2, 162.3, 163.4, 164.5, 165.6, 166.7, 167.8, 168.9, 170.0, 171.1, 172.2, 173.3, 174.4, 175.5, 176.6, 177.7, 178.8, 179.9, 181.0, 182.1, 183.2, 184.3, 185.4, 186.5, 187.6, 188.7, 189.8, 190.9, 192.0, 193.1, 194.2, 195.3, 196.4, 197.5, 198.6, 199.7, 200.8, 201.9, 203.0, 204.1, 205.2, 206.3, 207.4, 208.5, 209.6, 210.7, 211.8, 212.9, 214.0, 215.1, 216.2, 217.3, 218.4, 219.5, 220.6, 221.7, 222.8, 223.9, 225.0, 226.1, 227.2, 228.3, 229.4, 230.5, 231.6, 232.7, 233.8, 234.9, 236.0, 237.1, 238.2, 239.3, 240.4, 241.5, 242.6, 243.7, 244.8, 245.9, 247.0, 248.1, 249.2, 250.3, 251.4, 252.5, 253.6, 254.7, 255.8, 256.9, 258.0, 259.1, 260.2, 261.3, 262.4, 263.5, 264.6, 265.7, 266.8, 267.9, 269.0, 270.1, 271.2, 272.3, 273.4, 274.5, 275.6, 276.7, 277.8, 278.9, 280.0, 281.1, 282.2, 283.3, 284.4, 285.5, 286.6, 287.7, 288.8, 289.9, 291.0, 292.1, 293.2, 294.3, 295.4, 296.5, 297.6, 298.7, 299.8, 300.9, 302.0, 303.1, 304.2, 305.3, 306.4, 307.5, 308.6, 309.7, 310.8, 311.9, 313.0, 314.1, 315.2, 316.3, 317.4, 318.5, 319.6, 320.7, 321.8, 322.9, 324.0, 325.1, 326.2, 327.3, 328.4, 329.5, 330.6, 331.7, 332.8, 333.9, 335.0, 336.1, 337.2, 338.3, 339.4, 340.5, 341.6, 342.7, 343.8, 344.9, 346.0, 347.1, 348.2, 349.3, 350.4, 351.5, 352.6, 353.7, 354.8, 355.9, 357.0, 358.1, 359.2, 360.3, 361.4, 362.5, 363.6, 364.7, 365.8, 366.9, 368.0, 369.1, 370.2, 371.3, 372.4, 373.5, 374.6, 375.7, 376.8, 377.9, 379.0, 380.1, 381.2, 382.3, 383.4, 384.5, 385.6, 386.7, 387.8, 388.9, 390.0, 391.1, 392.2, 393.3, 394.4, 395.5, 396.6, 397.7, 398.8, 399.9, 401.0, 402.1, 403.2, 404.3, 405.4, 406.5, 407.6, 408.7, 409.8, 410.9, 412.0, 413.1, 414.2, 415.3, 416.4, 417.5, 418.6, 419.7, 420.8, 421.9, 423.0, 424.1, 425.2, 426.3, 427.4, 428.5, 429.6, 430.7, 431.8, 432.9, 434.0, 435.1, 436.2, 437.3, 438.4, 439.5, 440.6, 441.7, 442.8, 443.9, 445.0, 446.1, 447.2, 448.3, 449.4, 450.5, 451.6, 452.7, 453.8, 454.9, 456.0, 457.1, 458.2, 459.3, 460.4, 461.5, 462.6, 463.7, 464.8, 465.9, 467.0, 468.1, 469.2, 470.3, 471.4, 472.5, 473.6, 474.7, 475.8, 476.9, 478.0, 479.1, 480.2, 481.3, 482.4, 483.5, 484.6, 485.7, 486.8, 487.9, 489.0, 490.1, 491.2, 492.3, 493.4, 494.5, 495.6, 496.7, 497.8, 498.9, 500.0, 501.1, 502.2, 503.3, 504.4, 505.5, 506.6, 507.7, 508.8, 509.9, 511.0, 512.1, 513.2, 514.3, 515.4, 516.5, 517.6, 518.7, 519.8, 520.9, 522.0, 523.1, 524.2, 525.3, 526.4, 527.5, 528.6, 529.7, 530.8, 531.9, 533.0, 534.1, 535.2, 536.3, 537.4, 538.5, 539.6, 540.7, 541.8, 542.9, 544.0, 545.1, 546.2, 547.3, 548.4, 549.5, 550.6, 551.7, 552.8, 553.9, 555.0, 556.1, 557.2, 558.3, 559.4, 560.5, 561.6, 562.7, 563.8, 564.9, 566.0, 567.1, 568.2, 569.3, 570.4, 571.5, 572.6, 573.7, 574.8, 575.9, 577.0, 578.1, 579.2, 580.3, 581.4, 582.5, 583.6, 584.7, 585.8, 586.9, 588.0, 589.1, 590.2, 591.3, 592.4, 593.5, 594.6, 595.7, 596.8, 597.9, 599.0, 600.1, 601.2, 602.3, 603.4, 604.5, 605.6, 606.7, 607.8, 608.9, 610.0, 611.1, 612.2, 613.3, 614.4, 615.5, 616.6, 617.7, 618.8, 619.9, 621.0, 622.1, 623.2, 624.3, 625.4, 626.5, 627.6, 628.7, 629.8, 630.9, 632.0, 633.1, 634.2, 635.3, 636.4, 637.5, 638.6, 639.7, 640.8, 641.9, 643.0, 644.1, 645.2, 646.3, 647.4, 648.5, 649.6, 650.7, 651.8, 652.9, 654.0, 655.1, 656.2, 657.3, 658.4, 659.5, 660.6, 661.7, 662.8, 663.9, 665.0, 666.1, 667.2, 668.3, 669.4, 670.5, 671
```

Let's write some JSF

```
@Named
@RequestScoped
class Details {

    @Inject var log: Logger = null
    @Inject var model: Model = null
    @Inject var ctx: FacesContext = null
    @EJB var eventService: EventService = null
    @Inject var login: Login = null

    @BeanProperty var uid: String = null
    @BeanProperty var rating = -1
    @BeanProperty var ratingReadonly = false

    private var generalError: String = null

    @PostConstruct
    def init(): Unit = {
```

Event Details



Mama Mojo!

01. Mrz 2012, 18:00

Tarif	Price	Description	# available	Quantity
TAS	SFr. 23.04	Exchangeable, any row	details 653	<input type="text" value="0"/>
ERS	SFr. 17.94	Non-Exchangeable, any row	details 75	<input type="text" value="0"/>
T1	SFr. 20.35	Front three rows, non-refundable	details 45	<input type="text" value="0"/>
T2a	SFr. 15.95	Rows 4-10, non-refundable	details 3	<input type="text" value="0"/>

Tu quod Google nunc vertit ad et a Latine? Ago quod paucis tantum inveni, committitur admirans quod alia has hoc libro continet. Esset sortem.

[Add to cart](#)



```
<td colspan="2">
  <p:rating
    id="rater"
    value="#{details.rating}"
    readonly="#{details.ratingReadonly}"
    cancel="false"
  >
    <p:ajax event="rate" listener="#{details.doRating}" update="rater" />
    <p:ajax event="cancel" listener="#{details.doRating}" update="rater" />
  </p:rating>
</td>
```


Let's write a JPA entity

```
@Entity
@Access(AccessType.FIELD)
case class Rule(

  @Id
  @TableGenerator(name="RULE_GENERATOR", table="RULE_SEQUENCE", pkColumnValue="RULE")
  @GeneratedValue(strategy=GenerationType.TABLE, generator="RULE_GENERATOR")
  id: Long,

  name: String,
  expression: String,
  outcome: String,
  priority: Int,
  namespace: String,
  description: String
) {

  /** default no arg constructors are required by jpa */
  def this() = {
    this(0L, null, null, null, 0, null, null)
  }

  /** converts this rule into a rule engine rule */
  def convert() = new ch.maxant.rules.Rule(name, expression, outcome, priority, namespace, description)
}
```

Immutable, free #hashCode(), #equals(Object), #toString(), pattern matching, optimised copy constructor

Sidebar:

- Why is immutability interesting for persistence?
 - JDBC – required you to map from `ResultSet` to a class
 - Hibernate – no mapping... but change an attached object, and it results in an update SQL
 - Rule: Persistence layer objects may not be passed up to the business layer because we don't trust each other
 - Result: another mapper!
 - JPA has the “merge” method, to merge a modified copy of an immutable object into the DB. Scala supports lean copy constructors.
-

Let's write a JPA entity

```
@Entity
@Access(AccessType.FIELD)
case class Rule(

  @Id
  @TableGenerator(name="RULE_GENERATOR", table="RULE_SEQUENCE", pkColumnValue="RULE")
  @GeneratedValue(strategy=GenerationType.TABLE, generator="RULE_GENERATOR")
  id: Long,

  name: String,
  expression: String,
  outcome: String,
  priority: Int,
  namespace: String,
  description: String
) {

  /** default no arg constructors are required by jpa */
  def this() = {
    this(0L, null, null, null, 0, null, null)
  }

  /** converts this rule into a rule engine rule */
  def convert() = new ch.maxant.rules.Rule(name, expression, outcome, priority, namespace, description)
}
```

Immutable, free #hashCode(), #equals(Object), #toString(), pattern matching, optimised copy constructor

Let's write a JPA entity

```
@Entity
@Access(AccessType.FIELD)
@ScalaSignature(bytes="")
public class Rule
    implements Product, Serializable
{
    private final long id;
    private final String name;
    private final String expression;
    private final String outcome;
    private final int priority;
    private final String namespace;
    private final String description;
```

Generated decompiled bytecode

Where are the annotations?

A “case” class has no
bean-style accessor methods

Let's write a JPA entity

```
@Entity
@Access(AccessType.FIELD)
case class Rule(

  //dont forget the prefix!
  @JPAHelp.Id
  @JPAHelp.TableGenerator(name="RULE_GENER
  @JPAHelp.GeneratedValue(strategy=Generat
  id: Long,

  name: String,
  expression: String,
  outcome: String,
  priority: Int,
  namespace: String,
  description: String
) {
```

```
object JPAHelp {
  type Id = javax.persistence.Id @field
  type TableGenerator = javax.persistence.TableGenerator @field
  type GeneratedValue = javax.persistence.GeneratedValue @field
  type Column = javax.persistence.Column @field
  type ManyToOne = javax.persistence.ManyToOne @field
  type OneToMany = javax.persistence.OneToMany @field
  type ManyToMany = javax.persistence.ManyToMany @field
  type OrderBy = javax.persistence.OrderBy @field
  type MapKeyJoinColumn = javax.persistence.MapKeyJoinColumn @field
  type OrderColumn = javax.persistence.OrderColumn @field
  type Cacheable = javax.persistence.Cacheable @field
  type JoinColumn = javax.persistence.JoinColumn @field
  type Version = javax.persistence.Version @field
}
```

More complex example

```
@TransactionAttribute(TransactionAttributeType.REQUIRED)
def updateRating(eventUid: String, rating: Int, userId: Long)= {
  //TODO replace with ScalaQuery!!
  val q1 = em.createQuery("""
UPDATE Rating r
SET
  r.numRatings = (SELECT r2.numRatings + 1 FROM Rating r2 WHERE r2.eventUid = :eventUid)
, r.sumRatings = (SELECT r3.sumRatings + :newRating FROM Rating r3 WHERE r3.eventUid = :eventUid)
WHERE
  r.eventUid = :eventUid
""")
  q1.setParameter("eventUid", eventUid)
  q1.setParameter("newRating", rating.toLong)
  q1.executeUpdate

  //now block the user from doing another rating
  em.persist(RatingUserMapping(-1, eventUid, userId, rating))

  getRating(eventUid)
}
```


More complex example

```
@TransactionAttribute(TransactionAttributeType.REQUIRED)
def updateRating(eventUid: String, rating: Int, userId: Long)= {
  //TODO replace with ScalaQuery!!
  val q1 = em.createQuery("""
UPDATE Rating r
SET
  r.numRatings = (SELECT r2.numRatings + 1 FROM Rating r2 WHERE r2.eventUid = :eventUid)
, r.sumRatings = (SELECT r3.sumRatings + :newRating FROM Rating r3 WHERE r3.eventUid = :eventUid)
WHERE
  r.eventUid = :eventUid
  """)
  q1.setParameter("eventUid", eventUid)
  q1.setParameter("newRating", rating.toLong)
  q1.executeUpdate

  //now block the user from doing another rating
  em.persist(RatingUserMapping(-1, eventUid, userId, rating))

  getRating(eventUid)
}
```

- Why use ORM?
 - Type safety?
 - JPA isn't really type safe, even with the Criteria API
-

More complex example

- Impedance mismatch
 - Why solve SQL with OO?
- Java has libraries like JOOQ

```
SELECT FIRST_NAME, LAST_NAME, COUNT(*)  
FROM AUTHOR  
JOIN BOOK ON AUTHOR.ID = BOOK.AUTHOR_ID  
WHERE LANGUAGE = 'DE'  
AND PUBLISHED > '2008-01-01'  
GROUP BY FIRST_NAME, LAST_NAME  
HAVING COUNT(*) > 5  
ORDER BY LAST_NAME ASC NULLS FIRST  
LIMIT 2  
OFFSET 1  
FOR UPDATE  
OF FIRST_NAME, LAST_NAME
```

```
create.select(FIRST_NAME, LAST_NAME, count())  
  .from(AUTHOR)  
  .join(BOOK).on(Author.ID.equal(Book.AUTHOR_ID))  
  .where(LANGUAGE.equal("DE"))  
  .and(PUBLISHED.greaterThan(parseDate("2008-01-01")))  
  .groupBy(FIRST_NAME, LAST_NAME)  
  .having(count().greaterThan(5))  
  .orderBy(LAST_NAME.asc().nullsFirst())  
  .limit(2)  
  .offset(1)  
  .forUpdate()  
  .of(FIRST_NAME, LAST_NAME)
```

Example taken from www.jooq.org

ScalaQuery

```
case class User(id: Long, email: String, password: String){
  def roles = Roles.where(_.userId === id).list
}

val Users = new Table[User]("USER"){
  def id = column[Long]("ID", O.PrimaryKey)
  def email = column[String]("EMAIL")
  def password = column[String]("PASSWORD")
  def * = id ~ email ~ password <> (User, User.unapply _)

  def roles = List("asdf", "FDSA")
}
```

```
//setup
Users.where(u => u.email === "jane@maxant.ch").delete

//insert
assert(1 == Users.insert(User(1, "jane@maxant.ch", "pswd")))
```

- Compile time checking and type safety
 - Abstraction at the SQL level rather than OO
 - Supports ORM too, to save you having to write mapping code
-

What to watch out for

- JSF and JPA work with Java collections
 - Implicit converters are available in Scala
 - Breaking changes in Scala
 - Historically there have been some (Collections API)
 - Deprecated stuff will be removed
 - Java should be more harsh like this but...
 - Debugging is special
 - “problems” view in Eclipse is your good friend
-



Questions?

